

# Alberi Binari di Ricerca

Moreno Marzolla  
Dip. di Scienze dell'Informazione  
Università di Bologna

marzolla@cs.unibo.it  
<http://www.moreno.marzolla.name/>

Original work Copyright © Alberto Montresor, University of Trento  
(<http://www.dit.unitn.it/~montreso/asd/index.shtml>)

Modifications Copyright © 2009, Moreno Marzolla, Università di Bologna  
(<http://www.moreno.marzolla.name/teaching/ASD2009/>)

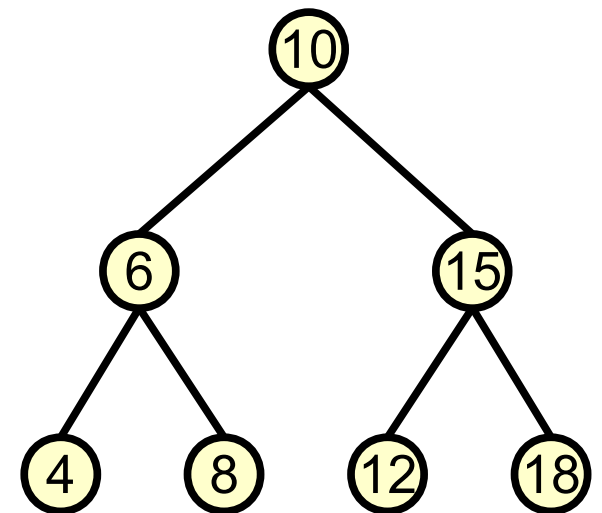
*This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.*

# Dizionari

- Dizionario
  - Insieme dinamico che implementa le funzionalità
    - Item search(Key key)
    - void insert(Key key, Item item)
    - void delete(Key key)
- Struttura dati fondamentale dalle infinite applicazioni
  - Esempio: individuare una tupla in un Database conoscendo la chiave di ricerca
- Esempi di implementazione
  - Array ordinato
    - Ricerca  $O(\log n)$ , inserimento/cancellazione  $O(n)$
  - Lista non ordinata
    - Ricerca/cancellazione  $O(n)$ , inserimento  $O(1)$

# Alberi binari di ricerca (ABR)

- Idea
  - Portare l'idea di ricerca binaria in un albero
- Definizione
  1. Ogni nodo  $v$  contiene un insieme di dati  $v.data$  associati ad una chiave  $v.key$  presa da un dominio totalmente ordinato (ci possono essere duplicati delle chiavi)
  2. Le chiavi dei nodi del sottoalbero sinistro di  $v$  sono  $\leq v.key$
  3. Le chiavi dei nodi del sottoalbero destro di  $v$  sono  $\geq v.key$



# Alberi binari di ricerca (ABR)

- Proprietà di ricerca
  - Le proprietà 2. e 3. permettono di realizzare un algoritmo di ricerca dicotomica
- **Domanda:** Proprietà di ordine
  - Come devo visitare l'albero per ottenere la lista ordinata dei valori?
- Dettagli implementativi
  - Ogni nodo deve mantenere
    - Figlio sinistro, destro
    - Padre
    - Chiave
    - Dati satelliti

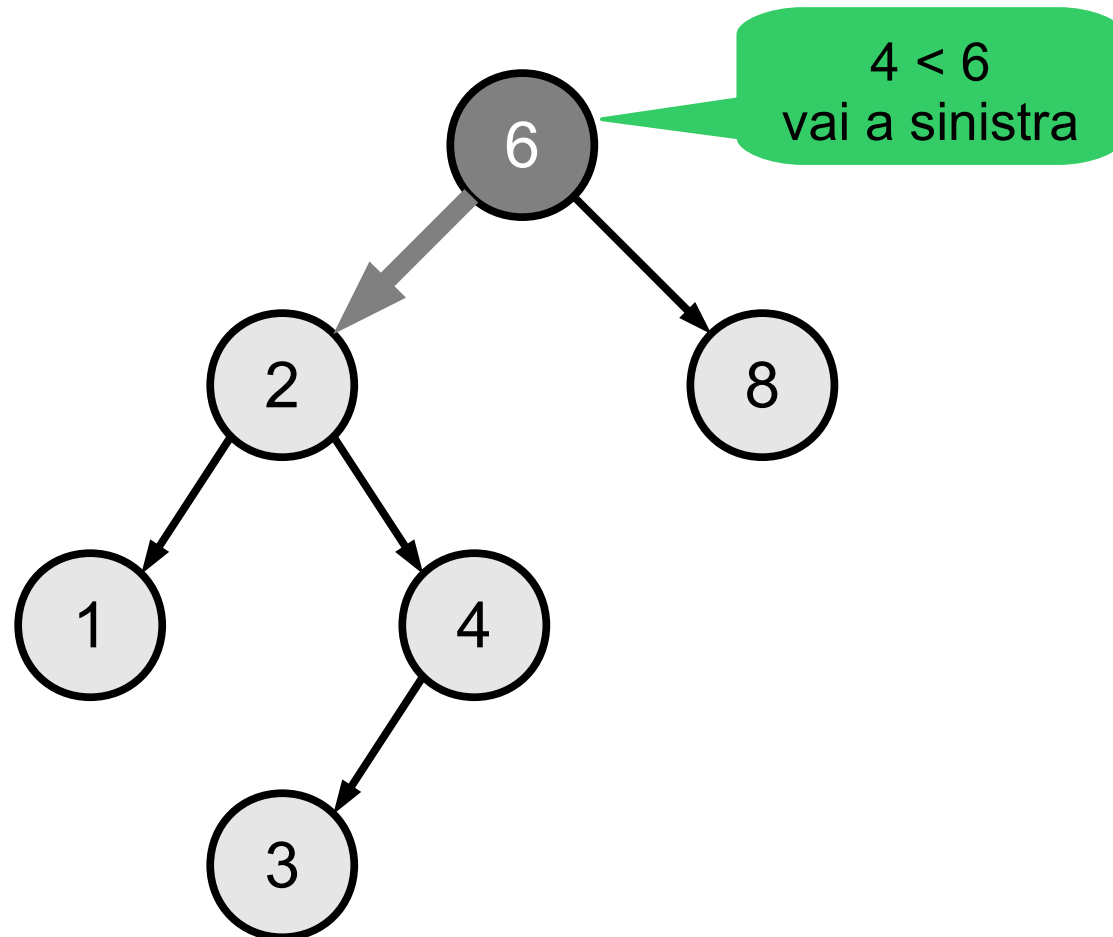


# Interfaccia Dizionario

```
public interface Dizionario {  
    /**  
     * Aggiunge al dizionario la coppia (e,k)  
     */  
    public Rif insert(Object e, Comparable k);  
  
    /**  
     * Rimuove dal dizionario l'elemento u  
     */  
    public void delete(Rif u);  
  
    /**  
     * Restituisce l'elemento <code>e</code> con chiave k.  
     * In caso di duplicati, l'elemento restituito  
     * e' scelto arbitrariamente tra quelli con chiave k.  
     */  
    public Object search(Comparable k);  
}
```

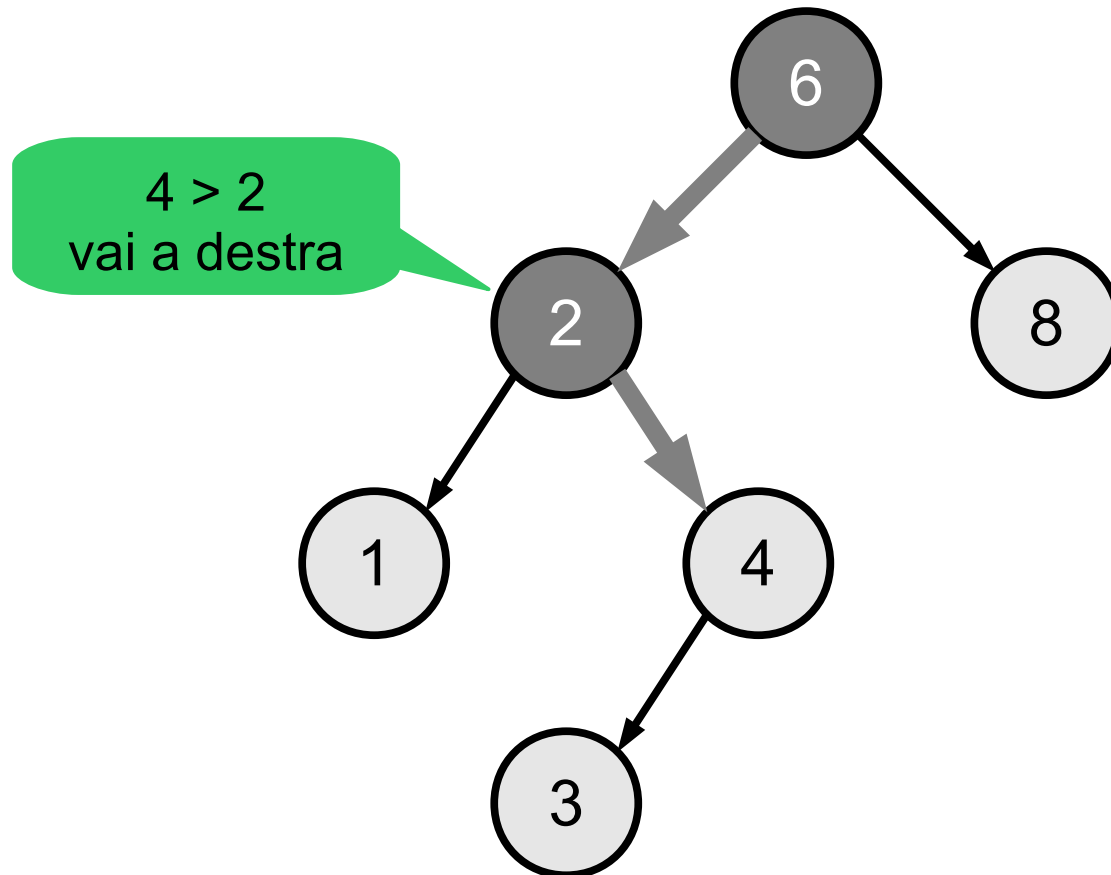
# Esempio

ricerca del valore 4



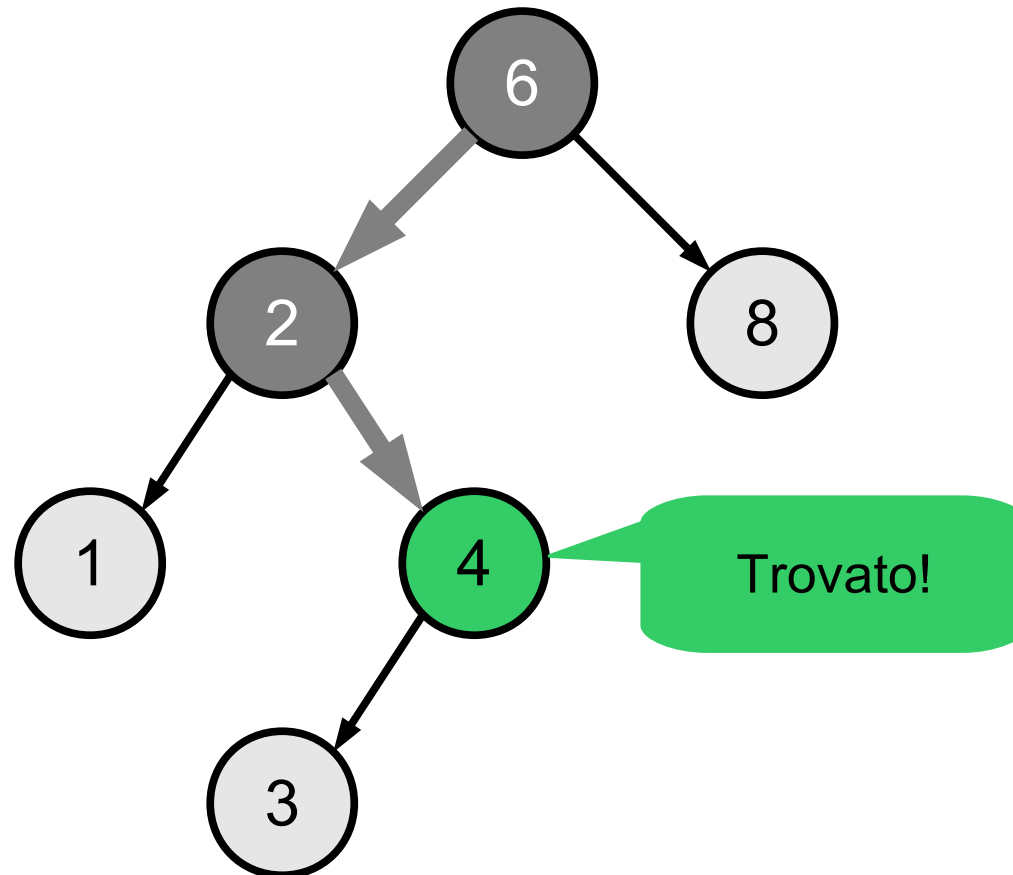
# Esempio

## ricerca del valore 4



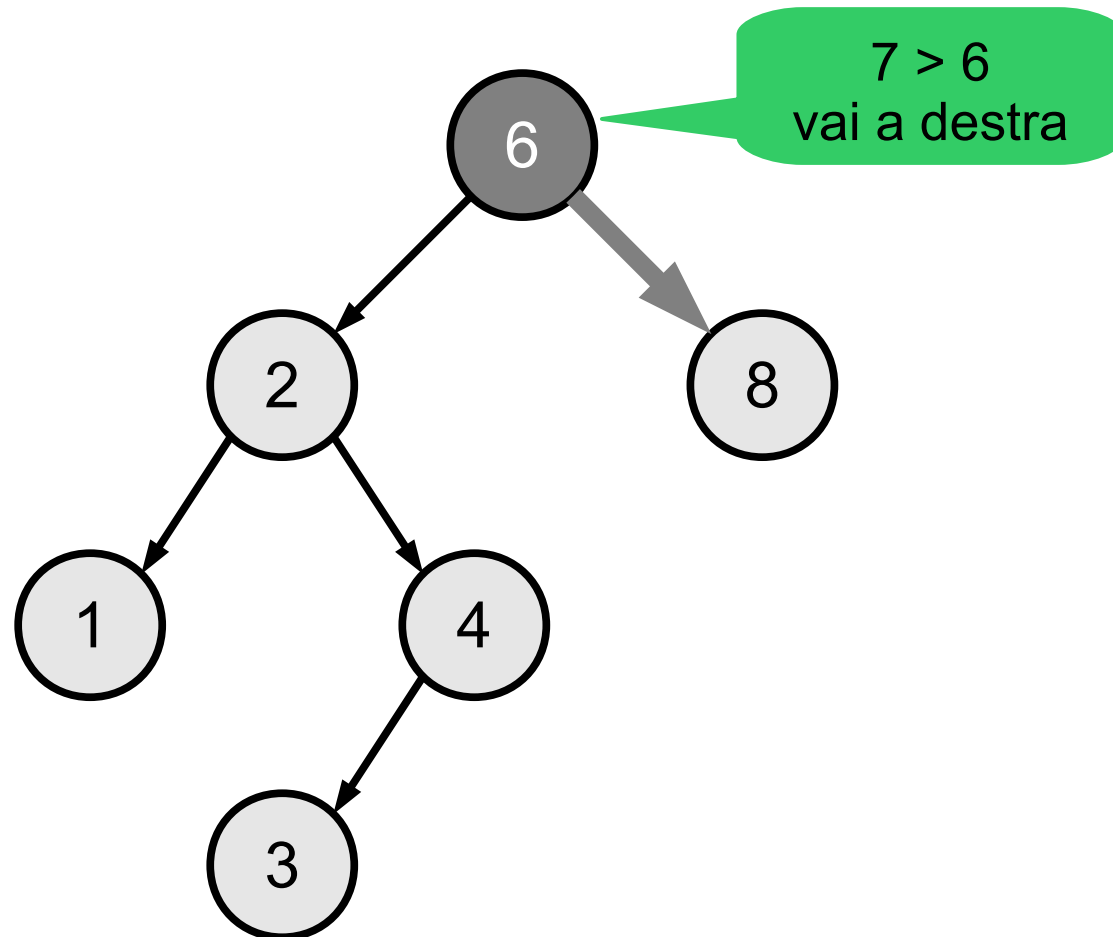
# Esempio

ricerca del valore 4



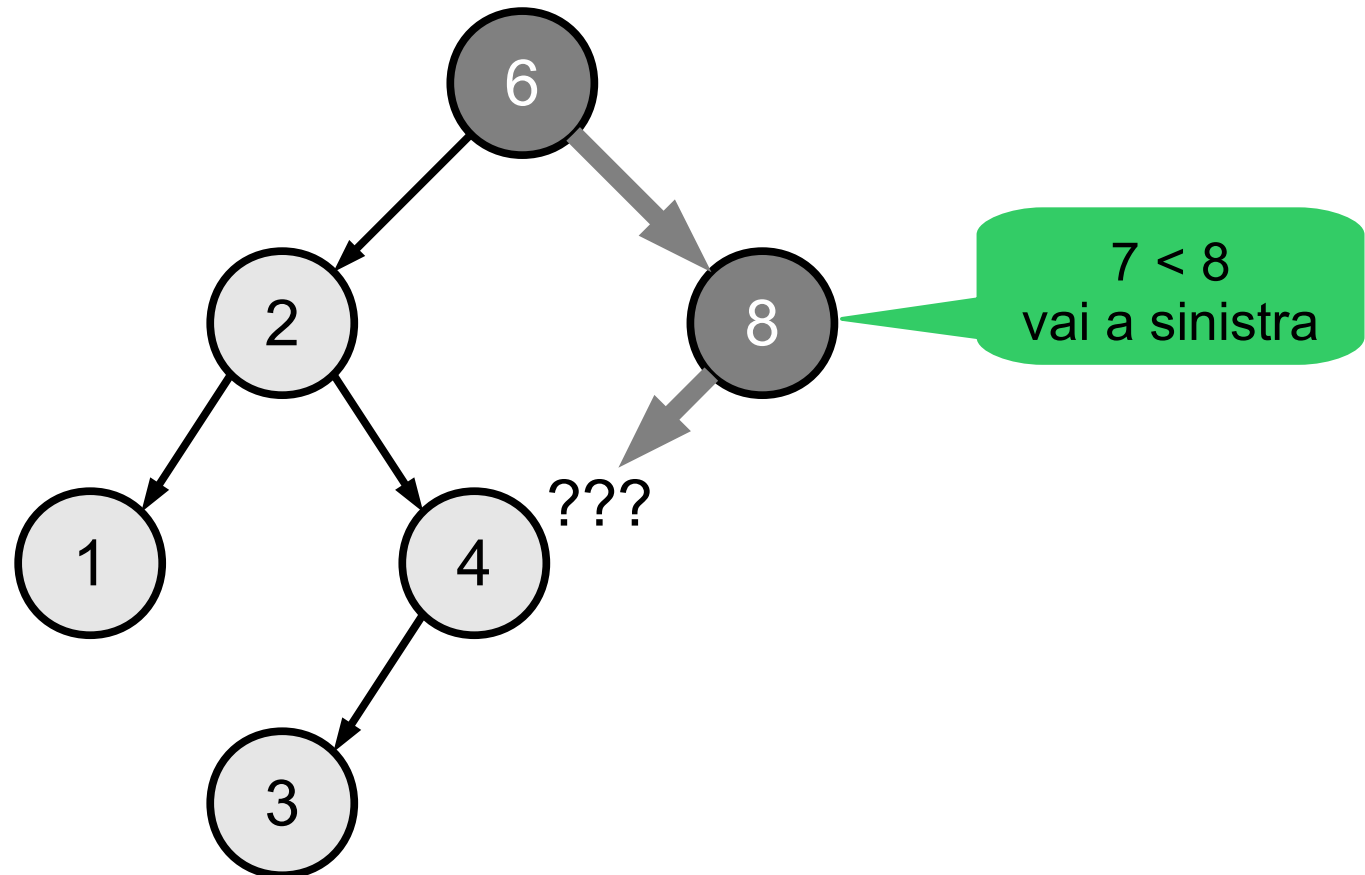
# Altro esempio

## ricerca del valore 7



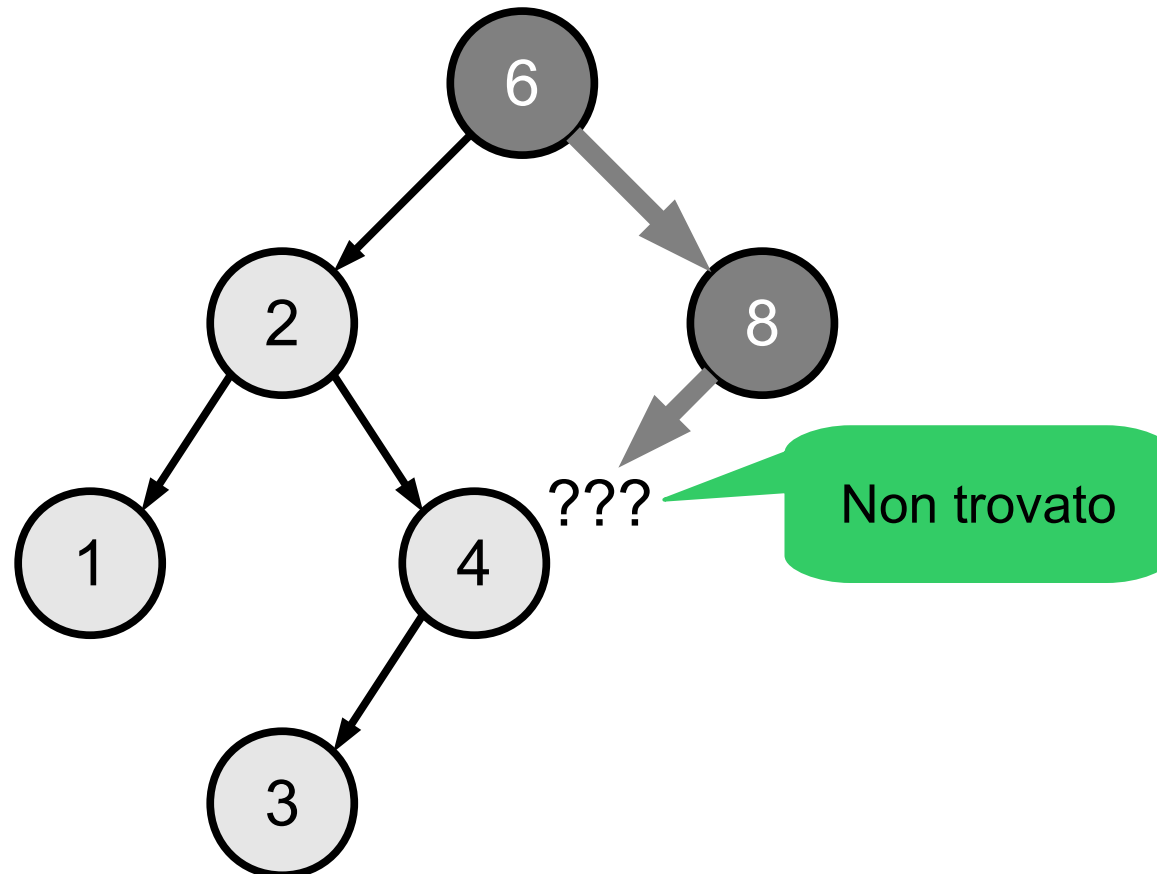
# Altro esempio

ricerca del valore 7



# Altro esempio

ricerca del valore 7



# Ricerca: pseudocodice

ABR search (ABR T, Key k)

```
if T = nil or k = T.key then
  return T
elseif k < T.key then
  return search(T.left, k)
else
  return search(T.right, k)
```

Versione  
ricorsiva

ABR search (ABR T, Key k)

```
while T ≠ nil do
  if k = T.key then
    return T
  elseif k < T.key then
    T = T.left
  else
    T = T.right
return nil
```

Versione  
iterativa

# Classe AlberoBR

package asdlab.libreria.AlberiRicerca

```
public class AlberoBR implements Dizionario {

    protected class InfoBR implements Rif {
        protected Object elem;
        protected Comparable chiave;
        protected Nodo nodo;
        protected InfoBR(Object e, Comparable k){
            elem = e; chiave = k; nodo = null;
        }
    }

    // Struttura dati "Albero Binario" in cui
    // sono memorizzate le informazioni
    protected AlberoBin alb;

    public AlberoBR() { ... }

    // altre operazioni ...
}
```

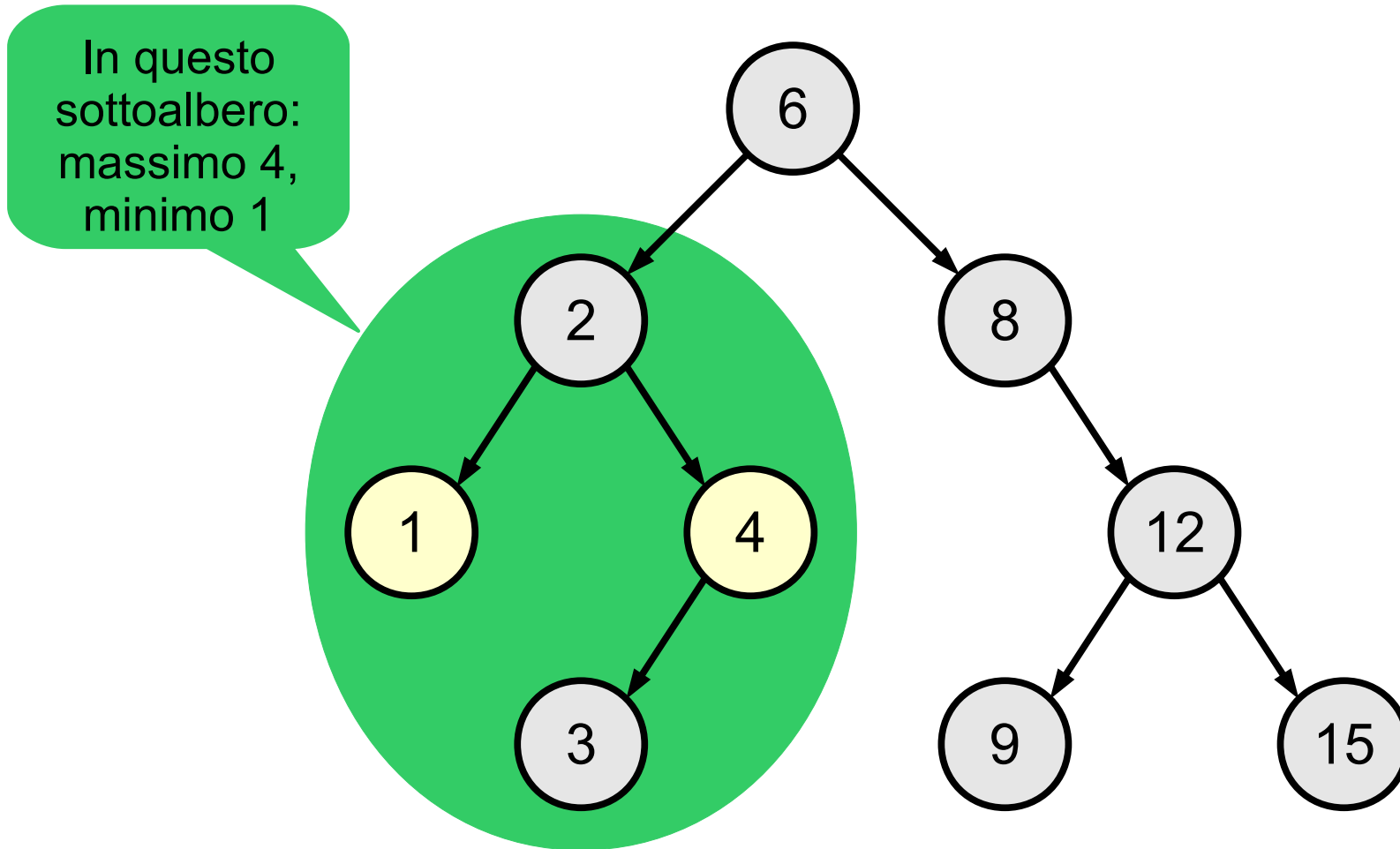
# Ricordiamo l'interfaccia AlberoBin (vedi slides su “Strutture Elementari”)

```
public interface AlberoBin {
    public enum TipoVisita { PREORDER, INORDER, POSTORDER };
    public int numNodi();
    public int grado(Nodo v);
    public Object info(Nodo v);
    public Nodo radice();
    public Nodo padre(Nodo v);
    public Nodo sin(Nodo v);
    public Nodo des(Nodo v);
    public void cambiaInfo(Nodo v, Object info);
    public Nodo aggiungiRadice(Object info);
    public Nodo aggiungiFiglioSin(Nodo u, Object info);
    public Nodo aggiungiFiglioDes(Nodo u, Object info);
    public void innestaSin(Nodo u, AlberoBin albero);
    public void innestaDes(Nodo u, AlberoBin albero);
    public AlberoBin pota(Nodo v);
    public List visitaDFS(TipoVisita t);
    public List visitaDFS();
    public List visitaBFS();
}
```

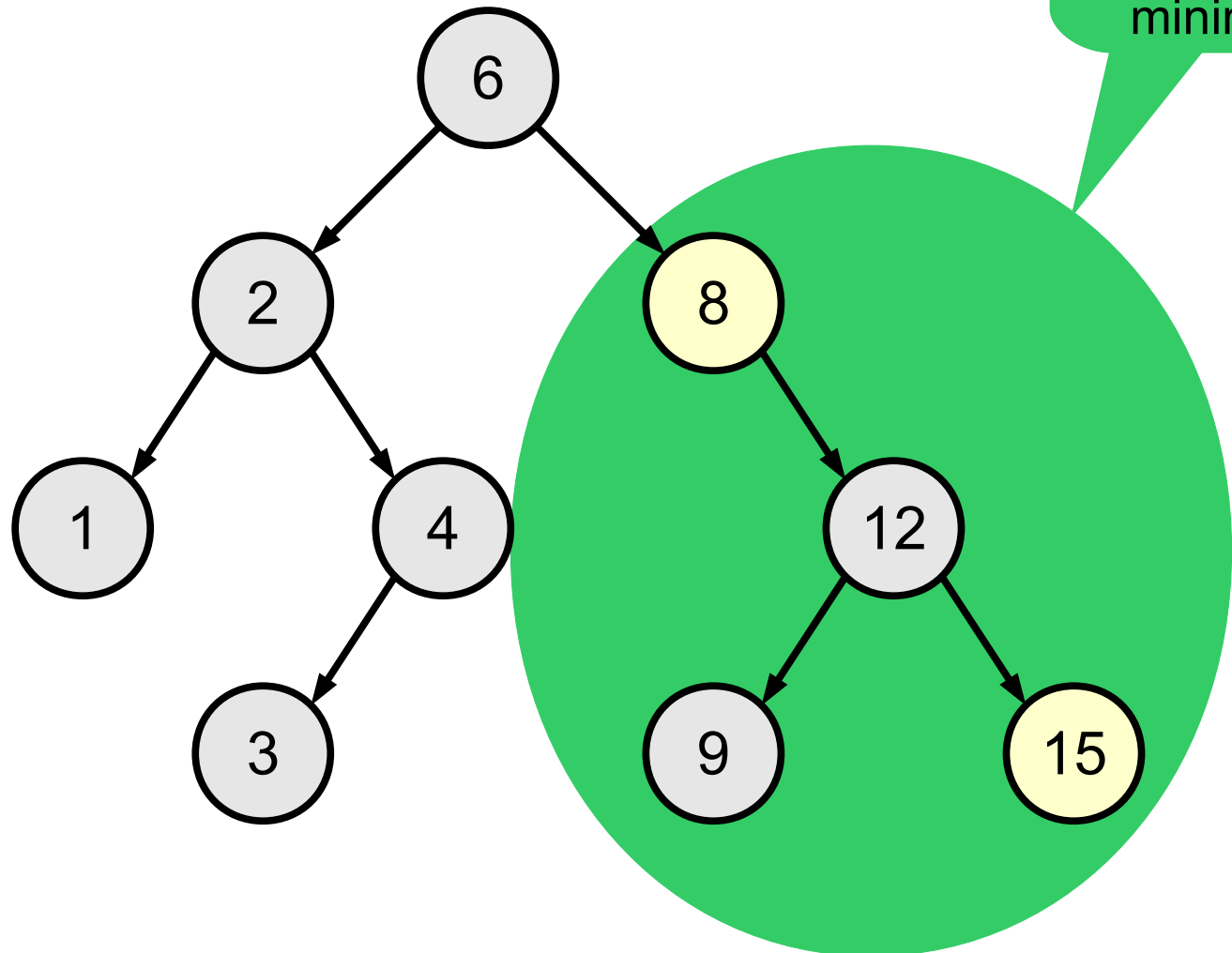
# Ricerca: implementazione Java (versione iterativa)

```
public Object search(Comparable k) {
    Nodo v = alb.radice();
    while (v != null) {
        InfoBR i = (InfoBR)alb.info(v);
        if (k.equals(i.chiave))
            return i.elem;
        if (k.compareTo(i.chiave) < 0)
            v = alb.sin(v);
        else
            v = alb.des(v);
    }
    return null;
}
```

# Minimo e Massimo



# Minimo e Massimo



# Ricerca del massimo

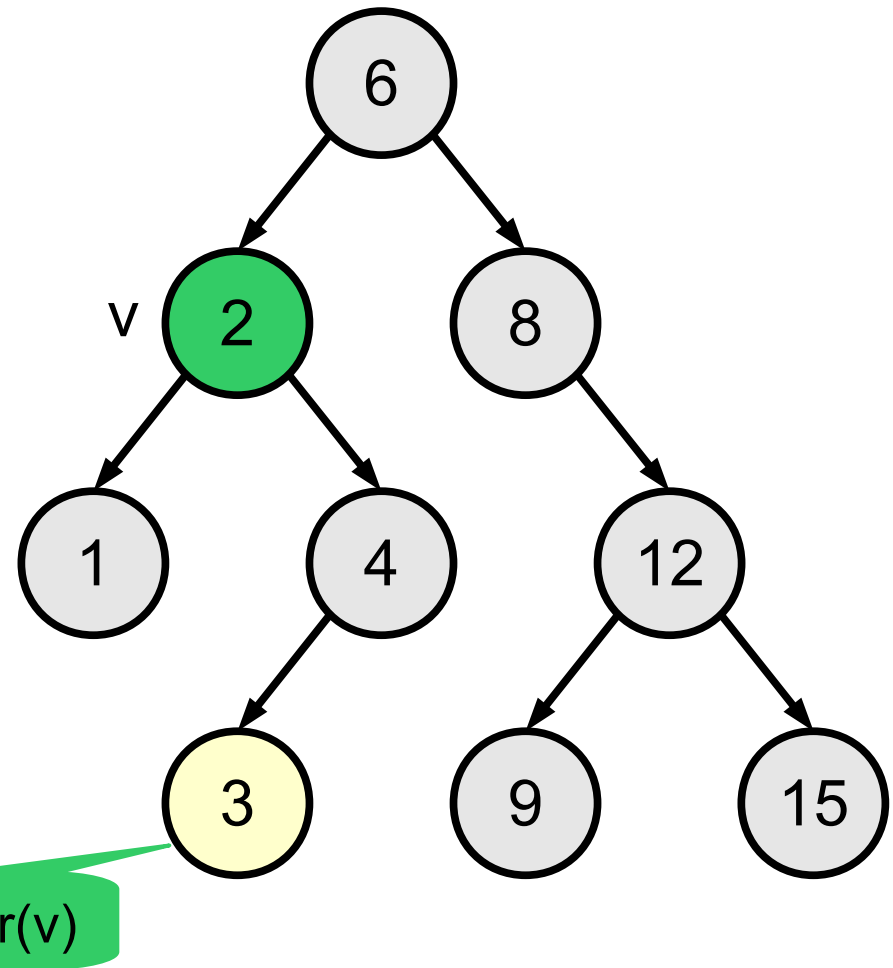
- Dato un nodo  $v$ :
  - Il valore **massimo** contenuto nell'albero con radice  $v$  è quello del nodo “più a destra”
  - Il valore **minimo** contenuto nell'albero con radice  $v$  è quello del nodo “più a sinistra”

```
protected Nodo max(Nodo v) {  
    while (v != null &&  
           alb.des(v) != null) {  
        v = alb.des(v);  
    }  
    return v;  
}
```

```
protected Nodo min(Nodo v) {  
    while (v != null &&  
           alb.sin(v) != null) {  
        v = alb.sin(v);  
    }  
    return v;  
}
```

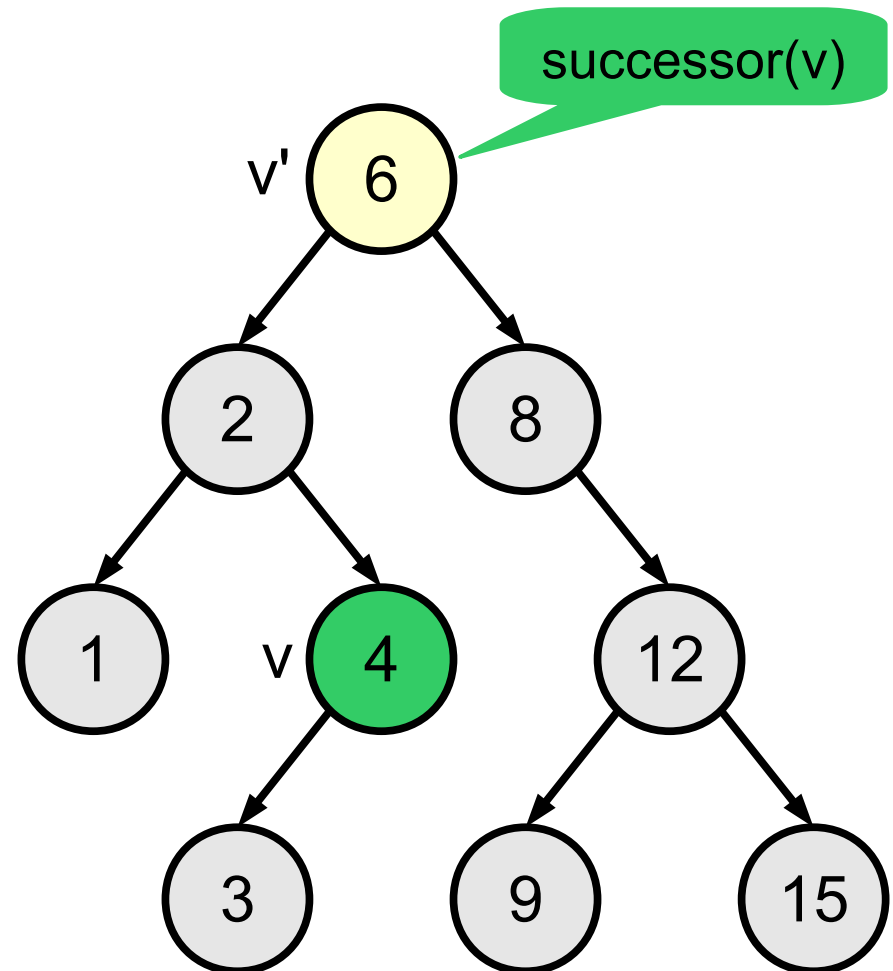
# Ricerca del successore

- Definizione
  - Il successore di un nodo  $v$  è il nodo contenente il più piccolo valore maggiore di  $v$
- Due casi
  - $v$  ha un figlio destro
  - Il successore è il minimo del sottoalbero destro di  $v$
  - Esempio:  
successore di 2 è 3



# Ricerca del successore

- Definizione
  - Il successore di un nodo  $v$  è il nodo contenente il più piccolo valore maggiore di  $v$
- Due casi
  - $v$  non ha un figlio destro
  - Il successore è il primo avo  $v'$  tale che  $v$  stia nel sottoalbero sinistro di  $v'$
  - Esempio:  
successore di 4 è 6

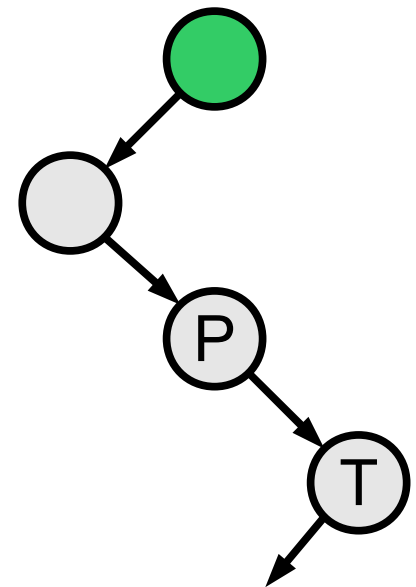


# Ricerca del successore Pseudo-codice (iterativo)

Caso 1: c'è  
un figlio  
destro

Caso 2: non  
c'è un figlio  
destro

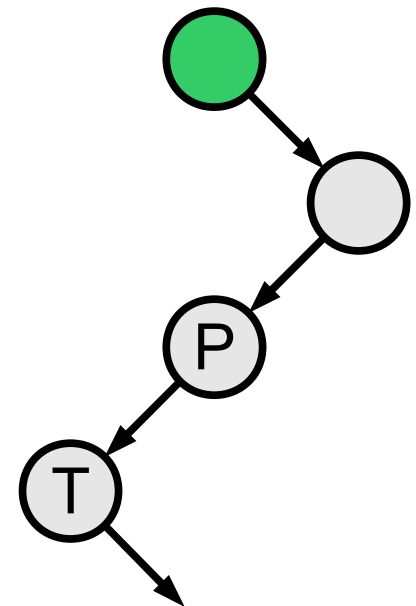
```
algorithm ABR successor (ABR T)
  if T = nil then
    return nil
  endif
  if (T.right ≠ nil) then
    return min(T.right)
  else
    P := T.parent
    while P ≠ nil and T = P.right do
      T := P
      P = P.parent
    endwhile
    return P
  endif
```



# Ricerca del predecessore

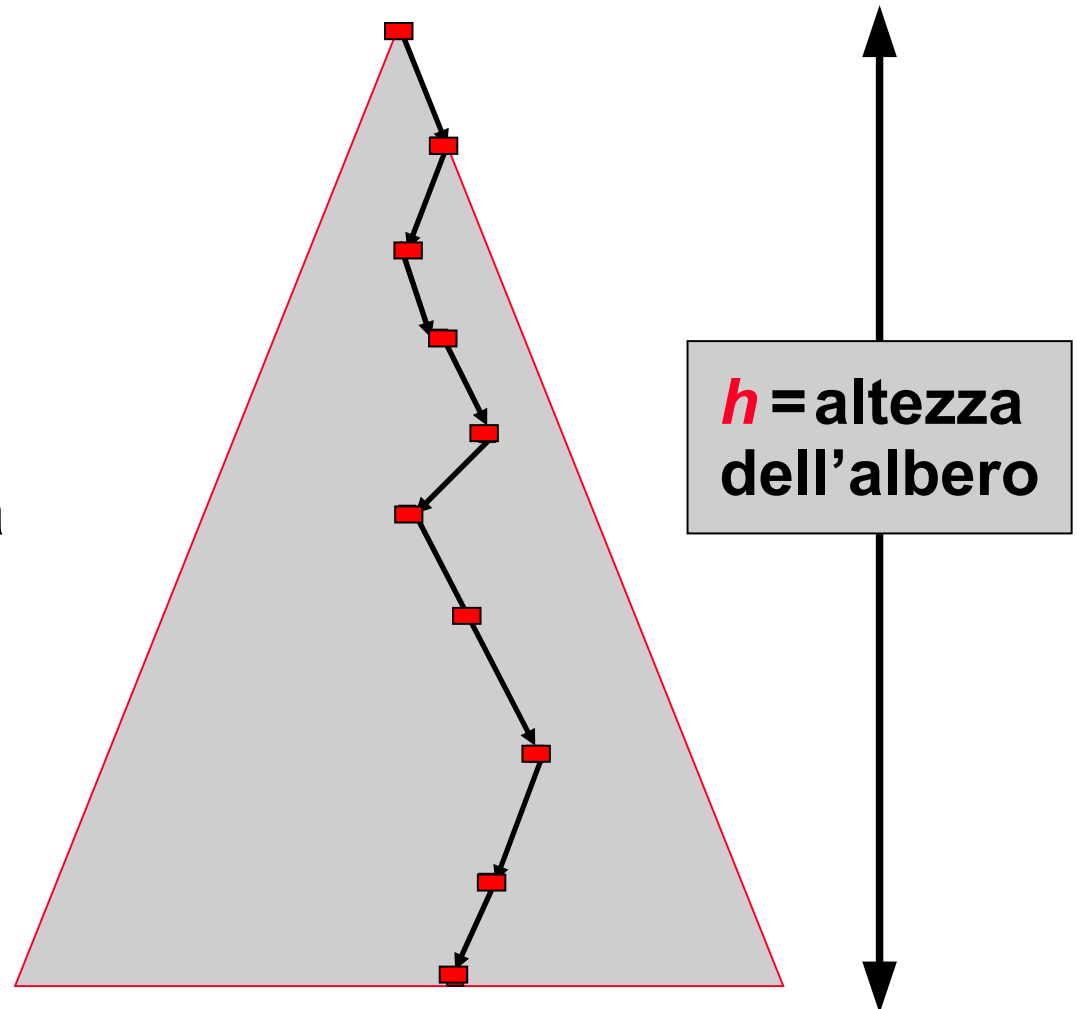
## Pseudo-codice (iterativo)

```
algorithm ABR predecessor (ABR T)
  if T = nil then
    return nil
  endif
  if (T.left ≠ nil) then
    return max(T.left)
  else
    P := T.parent
    while P ≠ nil and T = P.left do
      T := P
      P = P.parent
    endwhile
    return P
  endif
```



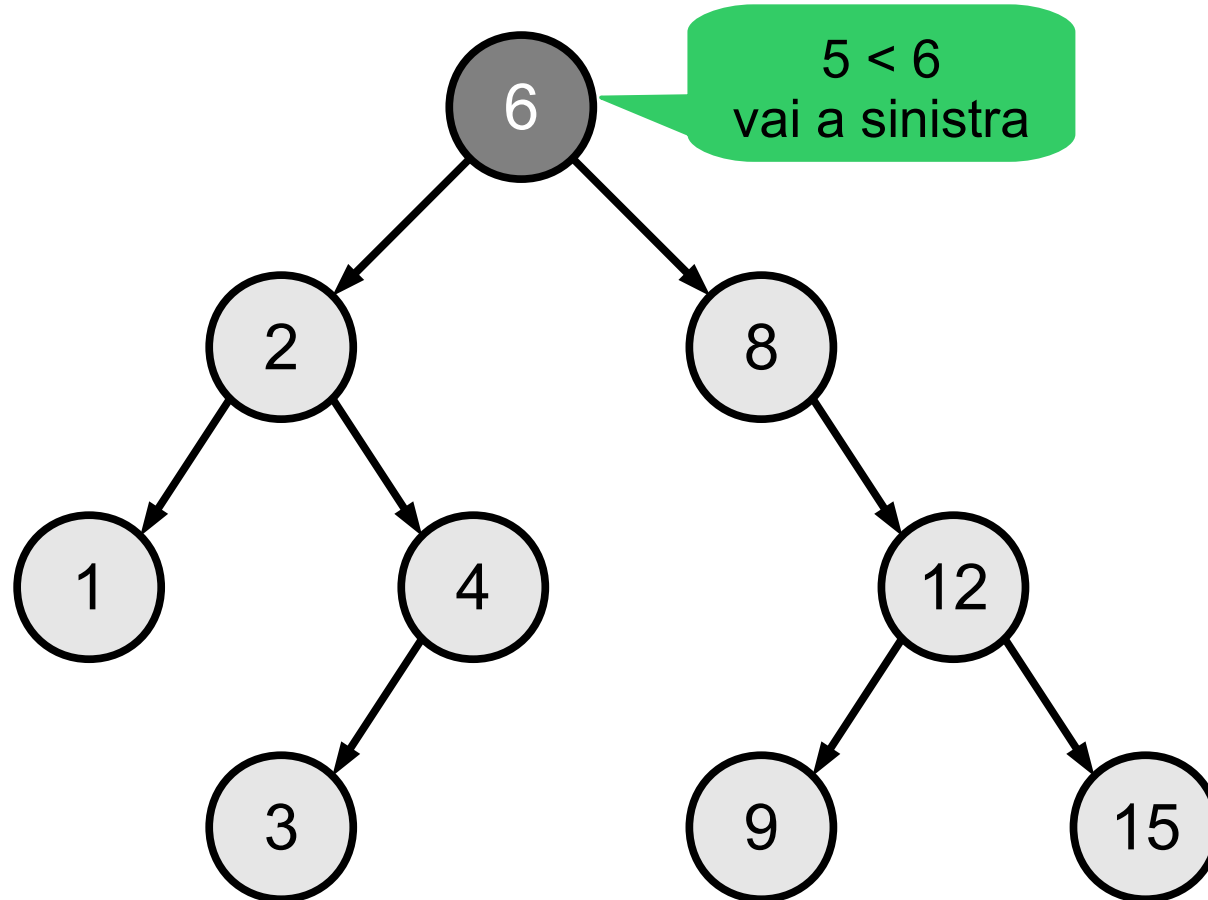
# Ricerca: costo computazionale

- In generale
  - Le operazioni di ricerca sono confinate ai nodi posizionati lungo un singolo percorso (path) dalla radice ad una foglia
  - Tempo di ricerca:  $O(h)$
- **Domanda**
  - Qual è il caso pessimo?
- **Domanda**
  - Qual è il caso ottimo?



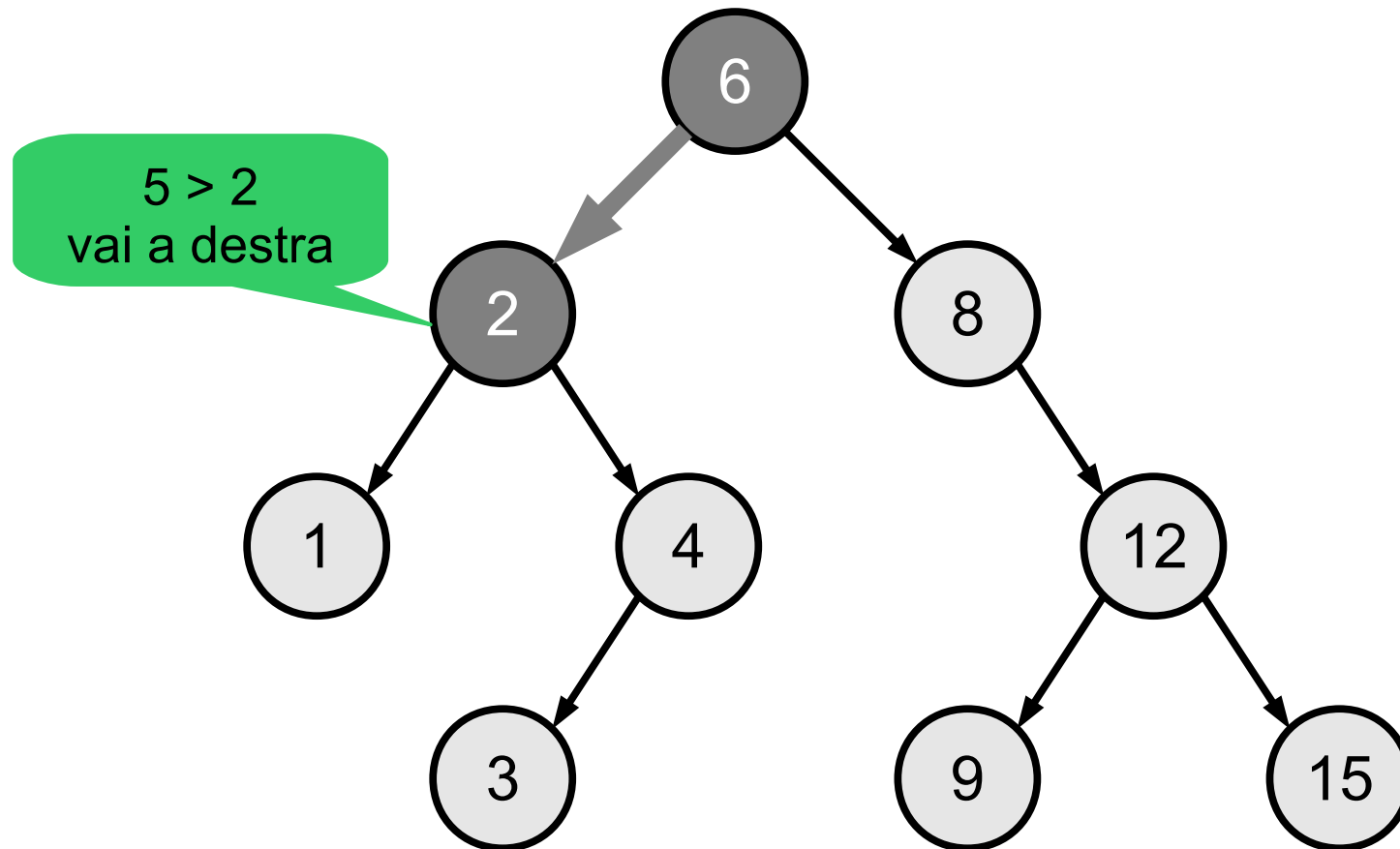
# Inserimento

## Inseriamo il valore 5



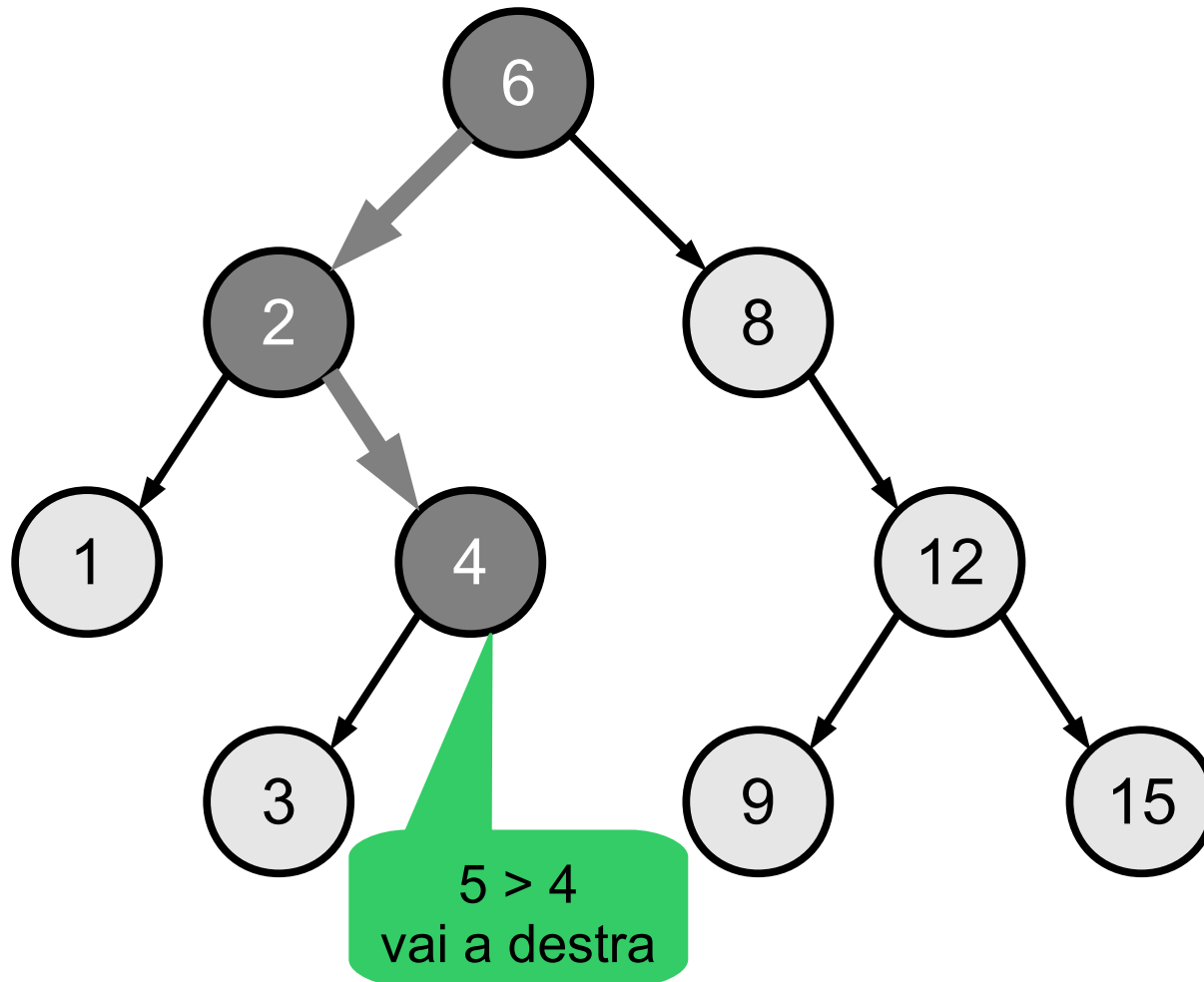
# Inserimento

## Inseriamo il valore 5



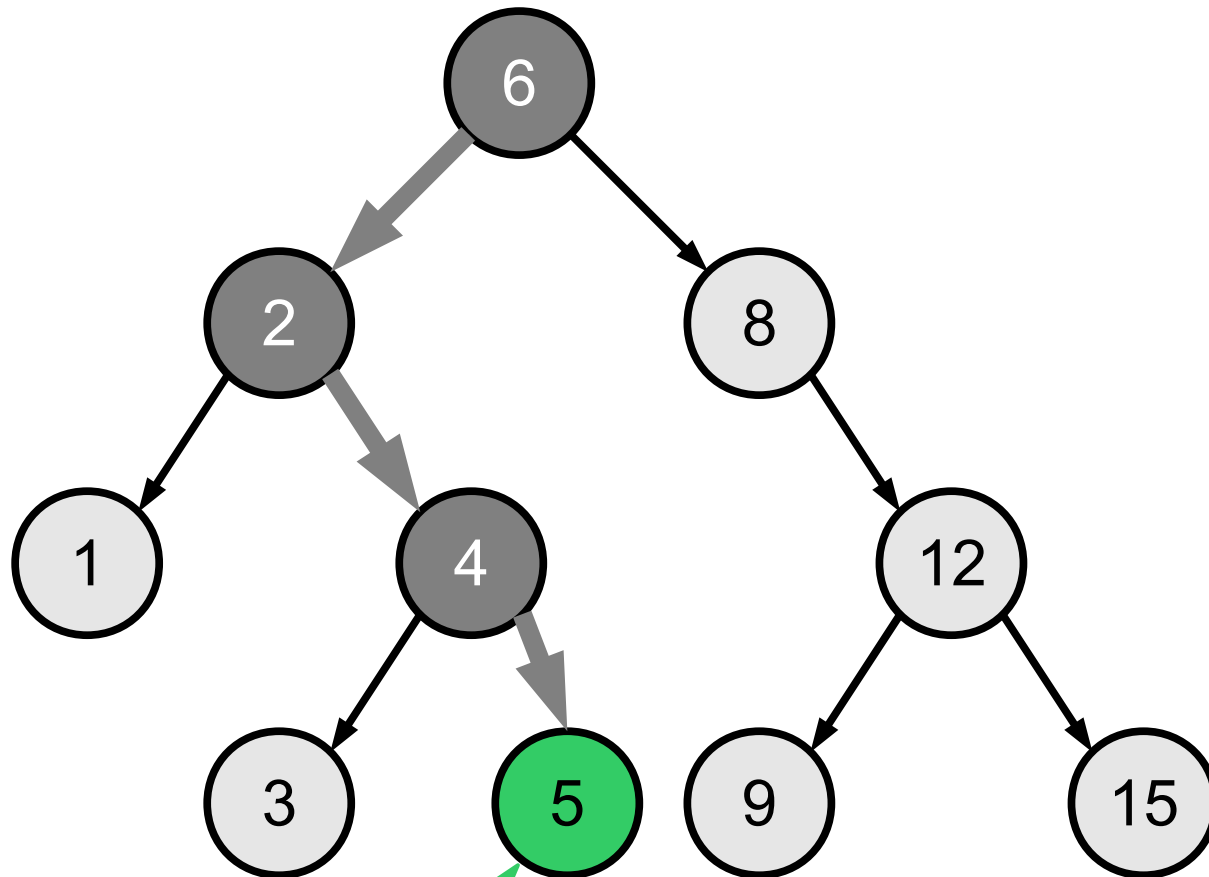
# Inserimento

## Inseriamo il valore 5



# Inserimento

Inseriamo il valore 5



Inserimento  
effettivo

# Inserimento: pseudo-codice (iterativo)

```
ABR insert(ABR T, Key k, Data d)
```

```
P := nil
```

```
while T != nil do  
  P = T  
  if T.key > key then  
    T = T.left  
  else  
    T = T.right
```

```
N := new ABR(k,d)
```

```
N.parent := P
```

```
if P = nil then  
  return N
```

```
if k < P.key then  
  P.left = N  
else  
  P.right = N
```

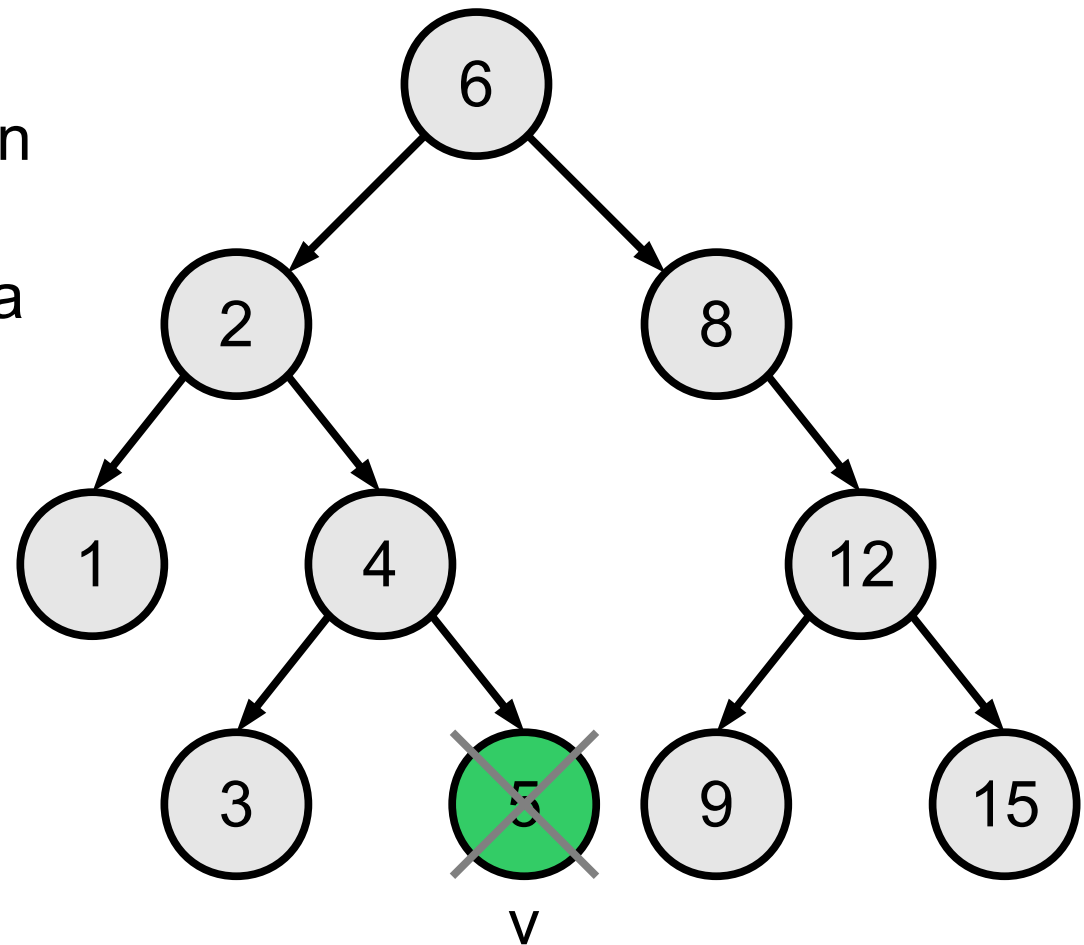
Ricerca posizione  
nuovo nodo

Caso base  
(inserimento su albero  
vuoto)

Caso generale

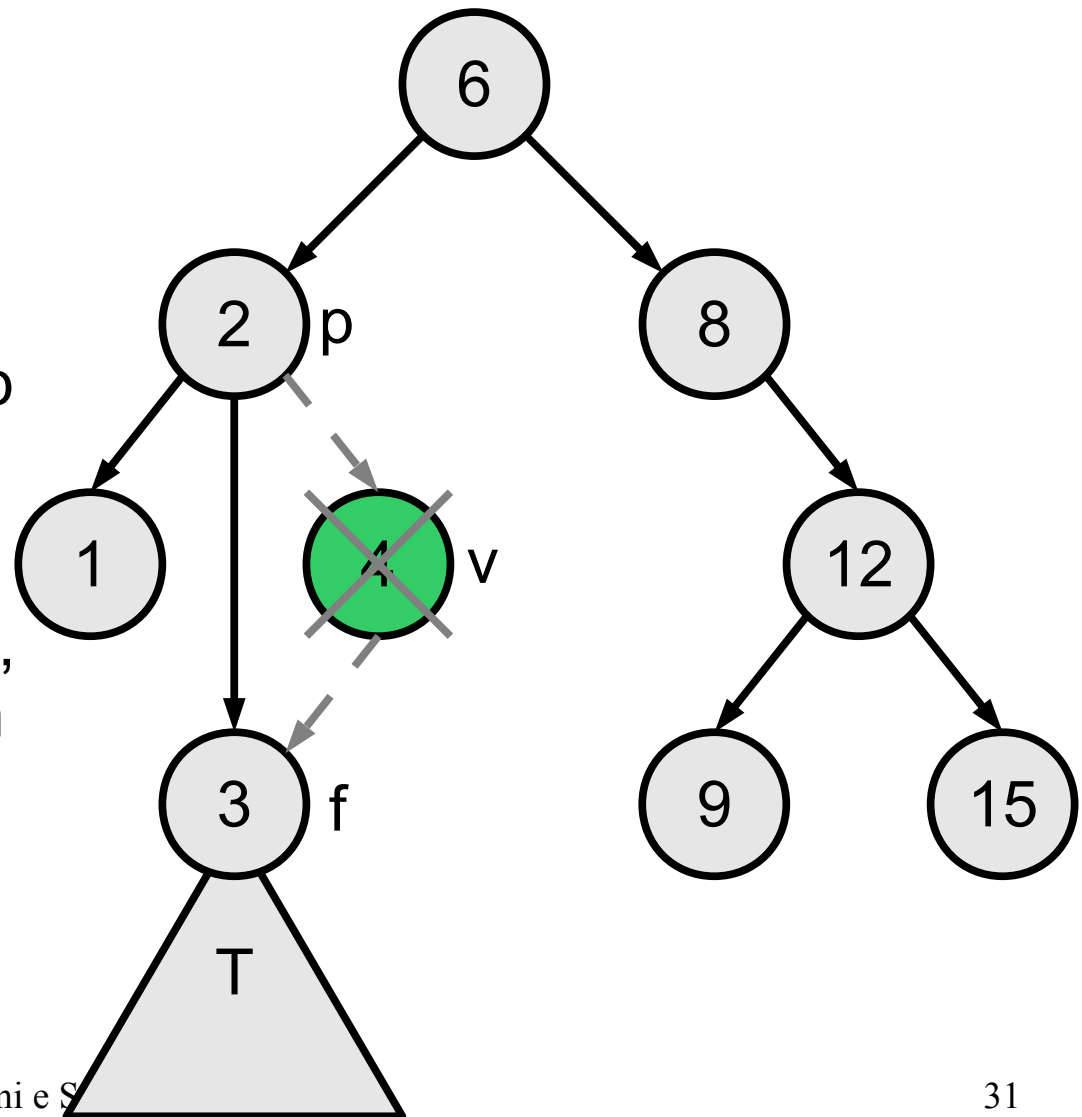
# Cancellazione

- Caso 1
  - Il nodo  $v$  da eliminare non ha figli
  - Semplicemente si elimina
- Correttezza
  - Eliminare una foglia non altera la proprietà di ordine degli altri nodi



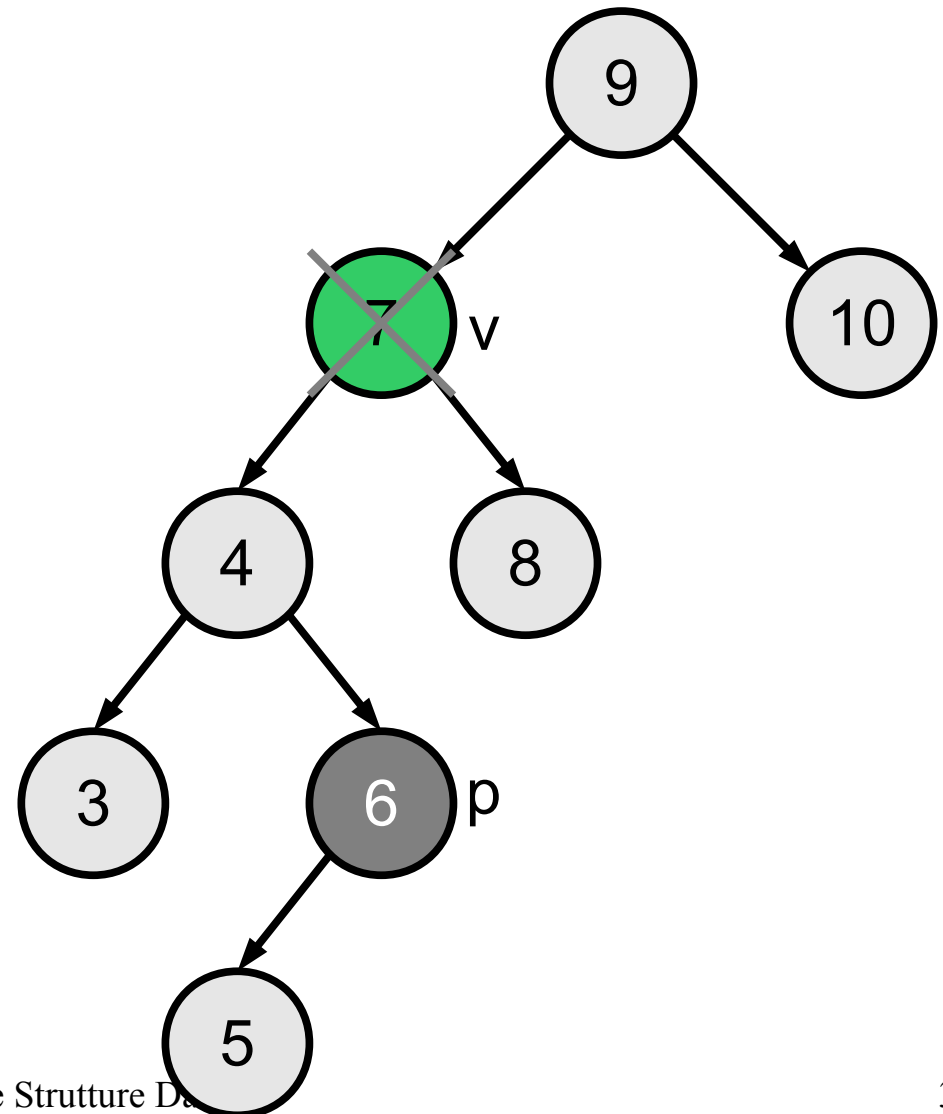
# Cancellazione

- Caso 2
  - Il nodo  $v$  da eliminare ha un unico figlio  $f$
  - Si elimina  $v$
  - Si attacca  $f$  all'ex-padre  $p$  di  $v$  in sostituzione di  $v$
- Correttezza
  - Per la proprietà di ordine, tutti i valori nelle chiavi in  $T$  sono  $\geq p$



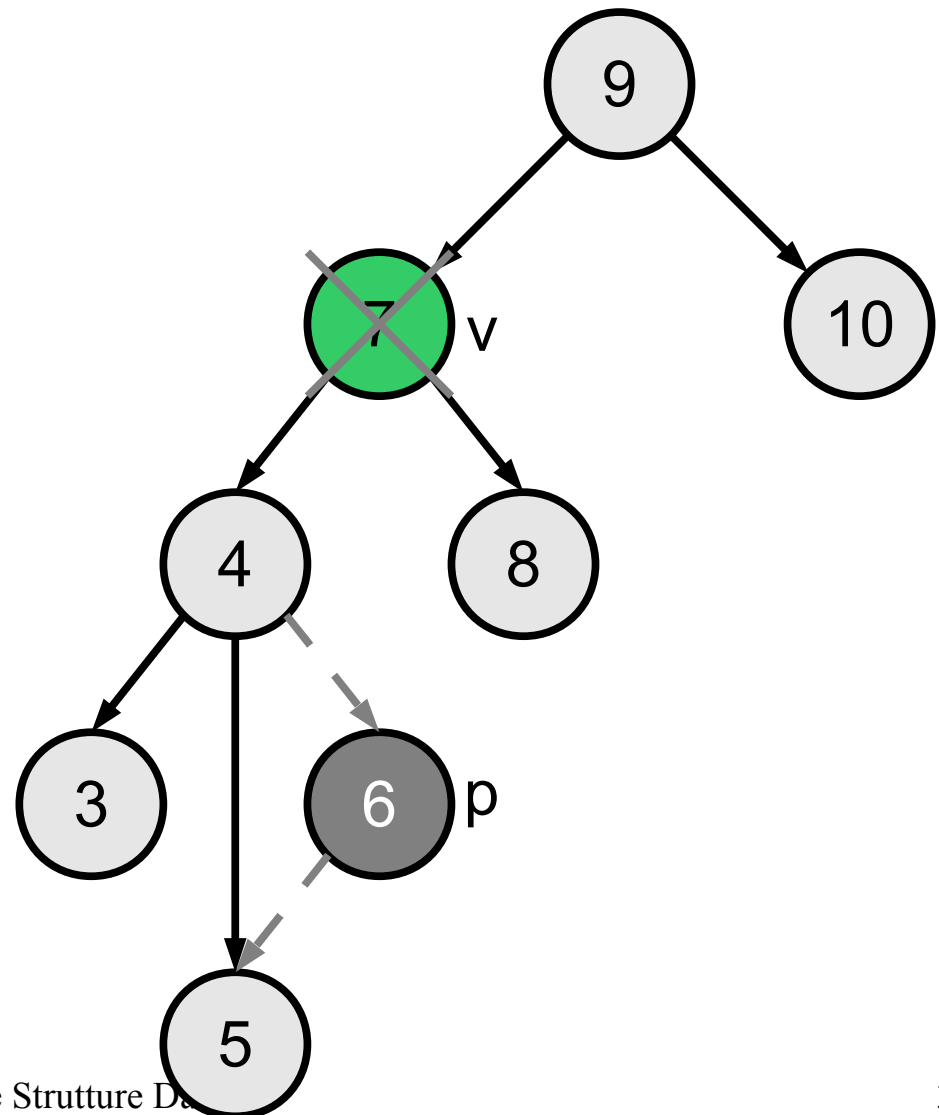
# Cancellazione

- Caso 3
  - Il nodo  $v$  da eliminare ha due figli
  - Si individua il predecessore  $p$  di  $v$
  - Il predecessore non ha figlio destro
    - Perché?



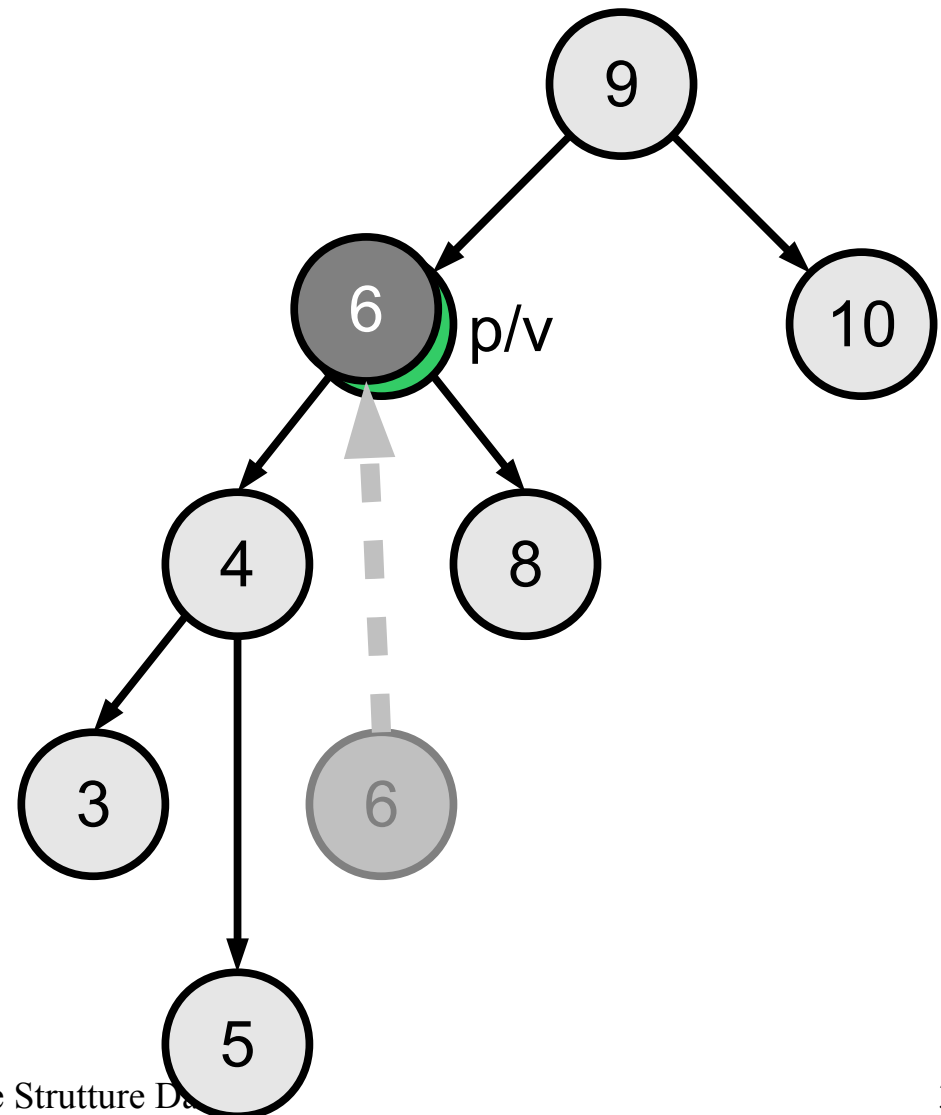
# Cancellazione

- Caso 3
  - Il nodo  $v$  da eliminare ha due figli
  - Si individua il predecessore  $p$  di  $v$
  - Il predecessore non ha figlio destro
  - Si “stacca” il predecessore
  - Si attacca l'eventuale figlio sinistro di  $p$  al padre di  $p$



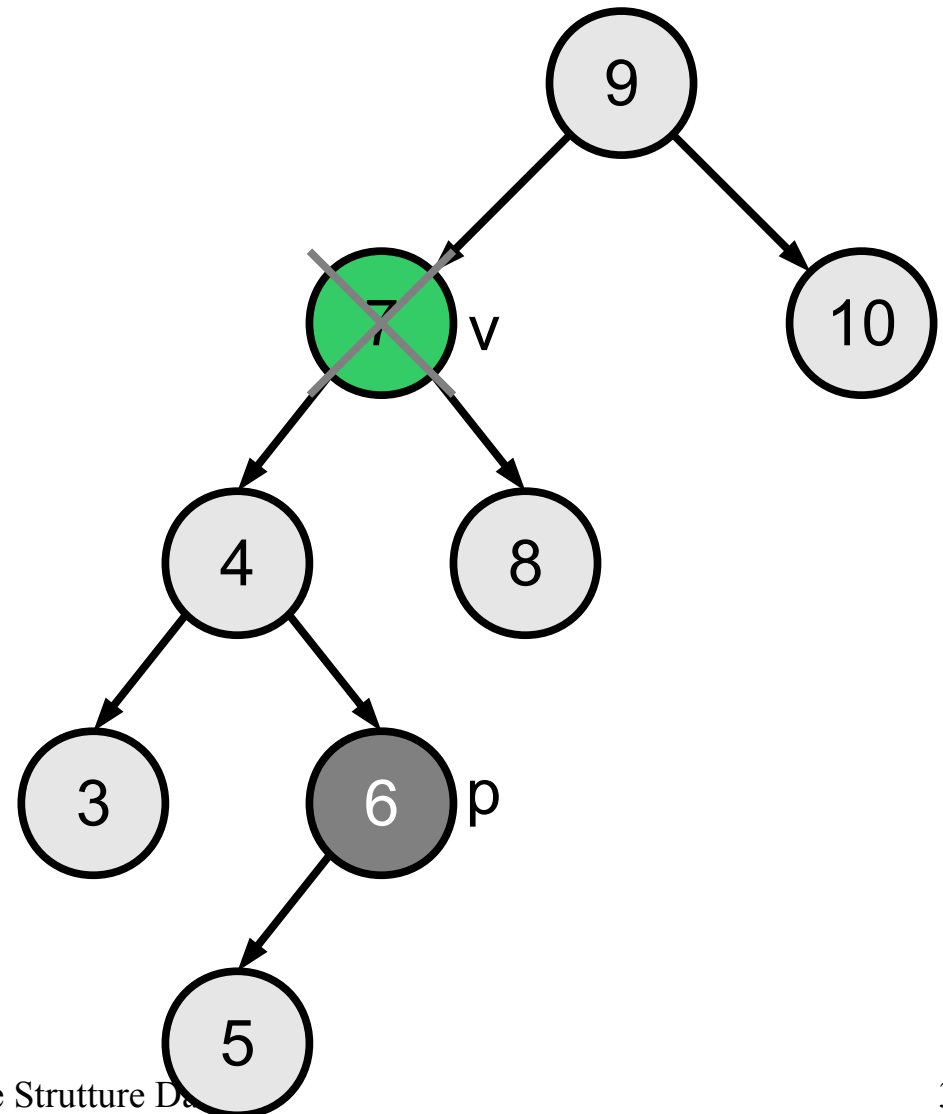
# Cancellazione

- Caso 3
  - Il nodo  $v$  da eliminare ha due figli
  - Si individua il predecessore  $p$  di  $v$
  - Il predecessore non ha figlio destro
  - Si “stacca” il predecessore
  - Si attacca l'eventuale figlio sinistro di  $p$  al padre di  $p$
  - Si copia  $p$  su  $v$



# Cancellazione

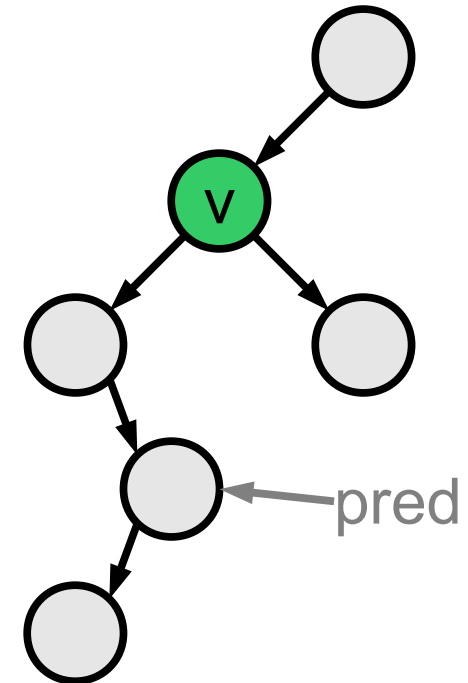
- Caso 3 (correttezza)
- Il predecessore  $p$  di  $v$ 
  - è sicuramente  $\geq$  di tutti i nodi del sottoalbero sinistro di  $v$
  - è sicuramente  $\leq$  di tutti i nodi del sottoalbero destro di  $v$
- Quindi può essere sostituito a  $v$



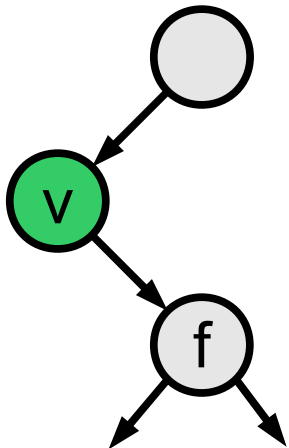
# Implementazione Java

```
protected Nodo delete(InfoBR i) {  
    Nodo v = i.nodo;  
    if (alb.grado(v) == 2) {  
        Nodo pred = max(alb.sin(v));  
        scambiaInfo(v, pred);  
        v = pred;  
    }  
    Nodo p = alb.padre(v);  
    contraiNodo(v);  
    return p;  
}
```

Elimina il nodo v  
(che può avere  
al più un figlio)



# Implementazione Java



```
protected void contraiNodo(Nodo v) {  
    Nodo f = null ;  
    if (alb.sin(v) != null)  
        f = alb.sin(v);  
    else  
        if (alb.des(v) != null)  
            f = alb.des(v);  
    if (f == null)  
        alb.pota(v);  
    else {  
        scambiaInfo(v, f);  
        AlberoBin a = alb.pota(f);  
        alb.innestaSin(v, a.pota(a.sin(f)));  
        alb.innestaDes(v, a.pota(a.des(f)));  
    }  
}
```

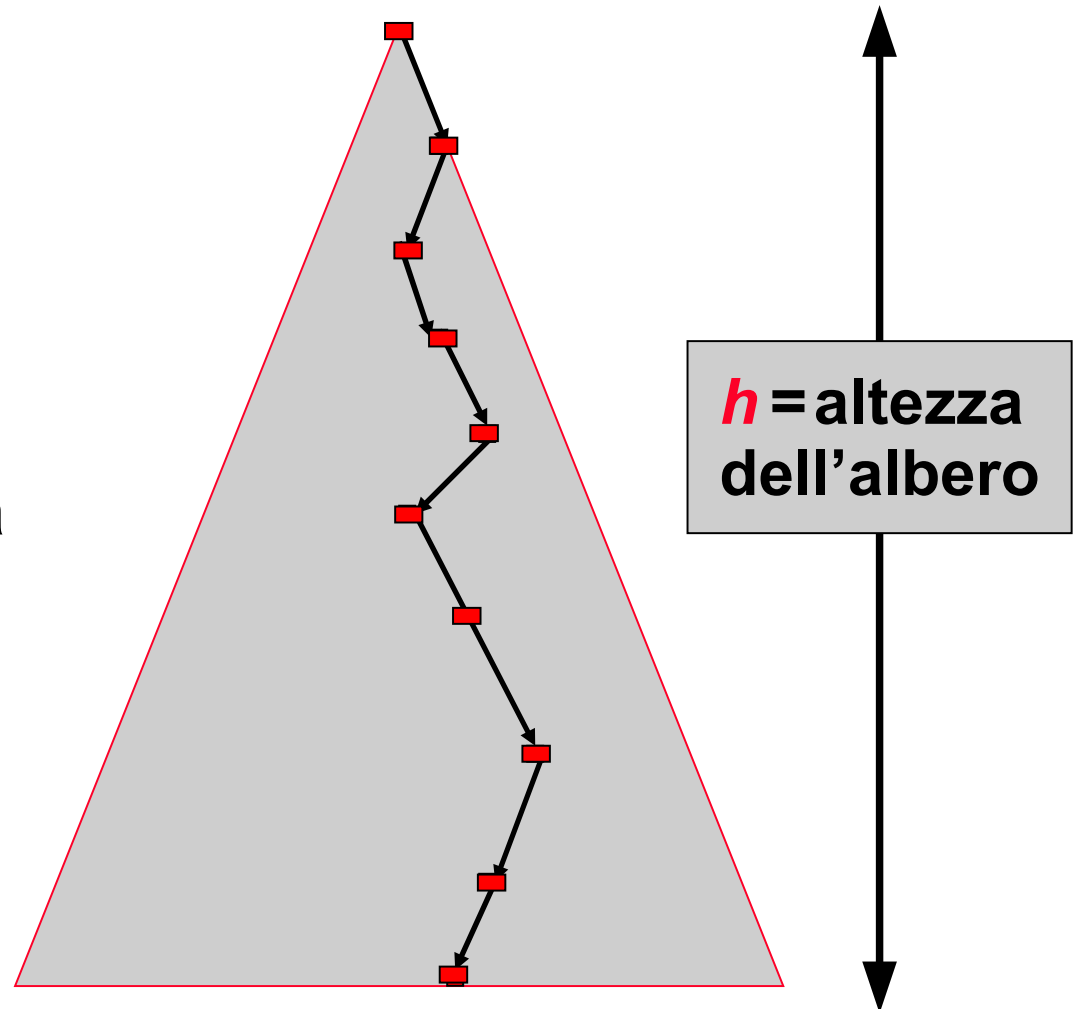
Elimina il nodo  $v$   
(che può avere  
al più un figlio)

$v$  non ha figli

stacca il  
sottoalbero  
radicato in  $f$

# Modifica: costo computazionale

- In generale
  - Le operazioni di modifica sono confinate ai nodi posizionati lungo un singolo percorso (path) dalla radice ad una foglia
  - Tempo:  $O(h)$



# Complessità media

- Qual è l'altezza media di un albero di ricerca?
  - Caso generale (inserimenti + cancellazione)
    - Difficile da trattare
  - Caso “semplice”: inserimenti in ordine casuale
    - E' possibile dimostrare che l'altezza media è  $O(\log n)$
- In generale:
  - Sono necessarie tecniche per mantenere bilanciato l'albero
  - Esempi
    - Alberi AVL (Adelson-Velsky, Landis)
    - Red-Black Tree
    - Splay Tree
  - Non cambiate canale 😊

# Esercizi

- **Esercizio**
  - Scrivere un algoritmo non ricorsivo che effettua un attraversamento in-ordine di un ABR
- **Esercizio**
  - Dimostrare: qualunque algoritmo basato su confronti per costruire un ABR è  $\Omega(n \log n)$ 
    - Suggerimento: poiché l'ordinamento basato su confronti di  $n$  elementi è  $\Omega(n \log n)$ , allora ...
- **Esercizio**
  - Dimostrate che se un nodo in un ABR ha due figli, allora il suo successore non ha un figlio sinistro e il suo predecessore non ha un figlio destro

# Esercizi

- **Esercizio**
  - L'attraversamento di un ABR di  $n$  nodi può essere implementato trovando l'elemento minimo nell'ABR e poi chiamando  $n-1$  volte l'operazione `successor()`.
  - Dimostrate che questo algoritmo viene eseguito in tempo  $\Theta(n)$
- **Esercizio**
  - Scrivete una versione ricorsiva della procedura `insert()`
- **Esercizio**
  - L'operazione di cancellazione da un ABR è commutativa? Nel senso che cancellare prima  $x$  e poi  $y$ , oppure cancellare prima  $y$  e poi  $x$ , produce lo stesso ABR?